Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Blaž Malej

Delo z grafi v relacijskih in sodobnih nerelacijskih podatkovnih bazah

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Obilica podatkov v obliki grafov in omrežij je povzročila nastanek specializiranih podatkovnih baz, kot so npr. Neo4j, OrientDB in Oracle Spatial in GraphDB, pa tudi dodatnih shranjevalnih mehanizmov (storage engine) za relacijske sisteme, kot sta npr. OQGraph za MySQL/MariaDB in GraphpostgreSQL za PostgreSQL. Raziščite načine in jezike za poizvedovanje po grafih, ter primerjajte in ovrednotite izbrana orodja s praktično uporabo na realističnem problemu. Pri tem uporabite različne kriterije, kot so udobnost poizvedovanja, funkcionalnost, hitrost, podpora tipom problemov, učinkovita implementacija novih algoritmov, horizontalna skalabilnost.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Blaž Malej sem avtor diplomskega dela z naslovom:

Delo z grafi v relacijskih in sodobnih nerelacijskih podatkovnih bazah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 12. februarja 2016 Podpis avtorja:

Zahvaljujem se mentorju za pomoč in vodenje pri opravljanju diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
SUPB	database management system	sistem za upravljanje s podatkovnimi bazami
DBMS	database management system	sistem za upravljanje s podatkovnimi bazami
FOAF	friend of a friend problem	problem prijatelj prijatelja

Povzetek

V diplomskem delu smo se lotili raziskovanja grafnih podatkovnih baz, vse skupaj pa smo popestrili s primerjavo med grafnimi in relacijskimi podatkovnimi bazami. Grafne podatkovne baze so baze, ki so primarno namenjene za hranjenje podatkov o grafih in obdelavo le-teh. Graf (kot podatkovna struktura v računalniški znanosti) je množica vozlišč, ki so med seboj povezana s povezavami. Z grafi lahko učinkovito modeliramo različne primere iz sveta okoli nas, npr. socialno omrežje na Facebooku, pri čemer so osebe vozlišča, povezave pa prijateljstva med njimi.

Odločili smo se, da bomo delali z odprtokodnimi programi, za to smo naredili primerjavo med dvema trenutno najbolj popularnima bazama: grafno bazo Neo4j in relacijsko MySQL. Tema primerjav je bila hitrost izvajanja poizvedb, za primer poizvedbe pa smo si izbrali tipičen problem, ki se imenuje prijatelj prijatelja (angl. *friend of a friend*). Ta nastopi, ko hočemo poiskati vse prijatelje osebe v nekem poljubnem socialnem grafu.

Odločili smo se, da raziskovalni del diplomskega dela popestrimo z dodatnim grafnim SUPB. V našo primerjavo smo vključili še SUPB OrientDB, ki se pogosto pojavlja v primerjavah z Neo4j.

Ključne besede: Grafna podatkovna baza, Neo4j, MySQL, OrientDB.

Abstract

The thema of this diploma thesis are graph databases and the comparison between graph databases and relational databases. Graph databases are used to store and process all kinds of graph data. Graph, a data structure in computer science, is a set of vertices which are connected with edges. Using this principle, we can effectively model various real life examples, e.g. social network of Facebook, for which we could store in graph entities of friends as nodes, and friendships as edges.
We decided to compare two currently most popular open source databases, Neo4j - representing graph databases and a relational database named MySQL. We compared the execution time of typical query where one would want to get all the friends of a friend in arbitrary social network (also called friend of a friend problem). In our research we also compare another graph database named OrientDB, which is often used in various comparisons with Neo4j.

Keywords: Graph database, Neo4j, MySQL, OrientDB.

Poglavje 1

Uvod

Grafne podatkovne baze oz. grafni SUPB so se prvič pojavili z začetkom 21. stoletja in nastankom podjetij, kot so Google, Facebook, kasneje Twitter, itd. Naštetim podjetjem je skupno, da obvladujejo ogromne količine podatkov, katerih bistvo so povezave. Google je z "grafom spleta" shranil povezave med spletnimi stranmi, Facebook pa je ustvaril "graf prijateljstev". Podjetja so za te namene razvila svoje podatkovne baze, kasneje pa so prišle tudi prve grafne baze namenjene širši publiki. Ena iz med prvih takih je bila baza Neo4j, ki je nastala leta 2000, verzija 1.0 pa je bila izdana leta 2010 [1].

Seveda lahko grafne podatke shranimo v relacijsko bazo, vendar pa to pri veliki količini povezav ni tako učinkovito. Zaradi svoje narave, kjer so povezave entitete prvega reda, in povezanosti podatkov, grafne baze ohranjajo približno enako časovno učinkovitost poizvedb, ne glede na velikost podatkovne množice. To pa ne drži za relacijske baze, kjer moramo za poizvedbo uporabiti stik (angl. *join*). Stiki postajajo z večanjem podatkovne množice vedno počasnejši. V knjigi z naslovom *Neo4j in Action* smo zasledili navedbo, da bi naj bila baza Neo4j v določenih primerih tudi do milijonkrat hitrejša od MySQL [2]. Odločili smo se, da to navedbo tudi preverimo in za to uporabimo podoben primerjalni test. Tako, kot v literaturi smo tudi mi za primerjavo uporabili oba SUPB in algoritem iskanja prijateljev prijatelja, vendar pa smo naredili manjšo, a vseeno zelo pomembno spremembo. Av-

torji izbrane literature so SUPB Neo4j poganjali v t. i. vgrajenem načinu. To pomeni, da SUPB teče znotraj javanske aplikacije, posledično pa je seveda izvajanje poizvedb lahko zelo hitro, saj praktično ni nobenih režijskih stroškov za podatke, ki se pretakajo po mreži. Mi smo test opravili na način, ki je bolj običajen, torej smo iz uporabniške aplikacije dostopali do oddaljene baze. Za oddaljeni dostop uporablja Neo4j zahtevke HTTP, kar pomeni bistveno več časa, posvečenega režiji. Izkazalo se je, da Neo4j le ni tako zelo hitrejši od MySQL.

V diplomskem delu bomo termina grafna baza in grafni SUPB uporabljali kot sopomenki.

1.1 Struktura diplomskega dela

Diplomsko delo je sestavljeno iz dveh delov. V prvem delu si bomo grafne baze pogledali iz bolj teoretičnega vidika. Poglavje 2 predstavlja različne probleme, ki jih v računalništvu rešujemo z grafi, v poglavju 3 pa smo opisali različne tehnologije, zbrane pod imenom NoSQL. V poglavju 4 si bomo pogledali, kako delujejo grafni SUPB, in nekaj bolj popularnih sistemov tudi predstavili. Poglavju 5 je namenjeno predstavitvi relacijskih SUPB, opisali pa bomo tudi MySQL, ki smo ga uporabili v kasnejši primerjavi. V poglavju 6 bomo podrobneje opisali SUPB Neo4j in poizvedbeni jezik Cypher, s katerim bomo tvorili poizvedbe za primerjavo. S poglavjem 7 vstopimo v drugi del diplomskega dela, kjer smo opisali primerjavo SUPB in predstavili rezultate. Poglavje 8 je namenjeno opisu problema iskanja prijateljev prijatelja. V tem poglavju bomo opisali tudi podatke (graf filmskih igralcev), ki jih bomo uporabljali za testiranje poizvedb. Dobljene čase izvajanja poizvedb bomo opisali in slikovno predstavili v poglavju 9.

Poglavje 2

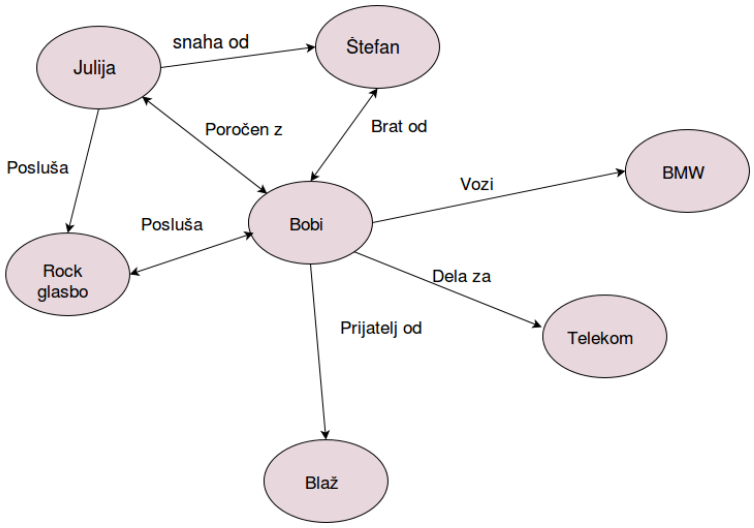
O grafih v računalništvu

Grafna podatkovna baza uporablja strukturo grafa za semantične poizvedbe (angl. *semantic queries*), osnovna gradnika grafa pa sta entiti vozlišče in povezava [3]. Če na graf pogledamo iz semantičnega vidika, vidimo, da vsako vozlišče predstavlja neko entiteto (to je lahko oseba, kraj, film ali neka poljubna stvar; npr. "oseba Janez"), vsaka povezava pa je zveza med dvema vozliščema (npr. "je poročen"). Vozliščem in povezavam lahko v obliki lastnosti in atributov dodamo dodatne informacije.

Na sliki 2.1 so vozlišča predstavljena kot krogci, v katerih so napisana imena entitet. Entitete so med seboj povezane s črtami, ki jim rečemo povezave (angl. *edge*), nad črto pa je napisan pomen te zveze. Iz takega grafa, kot je 2.1, je enostavno odgovoriti na vprašanja kot so: "Kateri avto vozi Bobi?", "S kom je poročena Julija?' ', "V katerem podjetju dela Bobi?". Graf, ki smo ga dali za primer, je zelo preprost, v realnosti pa so grafi po navadi veliko večji. Uporabljajo se na več področjih računalništva:

- Računalniška omrežja. Z grafi lahko predstavimo računalniška omrežja ali Internet. Računalniki, usmerjevalniki in druge naprave na omrežju predstavljajo vozlišča, fizične povezave med napravami pa predstavljajo povezave med vozlišči.

- Podatkovne strukture. Vsaka podatkovna struktura, ki uporablja kazalce, je v osnovi graf. Vse različne drevesne strukture in drevesa ali



Slika 2.1: Primer grafa, kjer smo modelirali neko skupino ljudi, odnose med njimi in druge lastnosti.

pa preprosti povezani seznami so v osnovi grafi.

- Socialna omrežja. Ogromna socialna omrežja, katerih glavni predstavnik je Facebook, so tipičen primer grafa, ki mu pravimo tudi socialni graf (angl. *social graph*). V Facebookovem grafu so vozlišča osebe, povezave pa predstavljajo prijateljstva. Prijateljstva so oboje-smerne povezave. Če je oseba A prijatelj osebe B sledi, da je tudi B prijatelj osebi A.

- Geografski podatki in iskanje poti. Da lahko najdemo najkrajšo možno pot med dvema točkama na zemljevidu (npr. Google Maps), bomo morali zemljevid modelirati kot graf in nad njem pognati algoritme za iskanje najkrajše poti (najbolj znan med njimi je Dijkstrov algoritem [31]).

Poglavje 3

Sodobni nerelacijski (NoSQL) SUPB

Če še do pred nekaj leti relacijski SUPB niso imeli pravega konkurenta, pa so ga v zadnjih nekaj letih dobili v gibanju NoSQL. Ogromne podatkovne baze, ki so postajale še večje in večje, so botrovale potrebi po učinkovitem skladiščenju teh podatkov. V družino NoSQL spada več med seboj konceptualno različnih si baz, med drugim tudi grafne baze, ki so tematika diplomskega dela.

Z besedo NoSQL označujemo vse SUPB, ki ne uporabljajo tradicionalnih relacijskih modelov. Naslednja podpoglavja so namenjena opisu različnih vej družine NoSQL, grafnim bazam pa smo namenili svoje poglavje.

3.1 SUPB tipa ključ-vrednost

Podatkovne baze ključ-vrednost so najpreprostejša oblika podatkovnih baz NoSQL. Iz njihovega imena je razvidno, kako delujejo. Osnova teh baz je podatkovna struktura slovarja (po navadi t. i. "hashmap"). To pomeni, da vsakemu podatku (vrednosti), ki smo ga vstavili v našo bazo, pripada unikaten ključ, ki ga uporabimo, ko potrebujemo ta podatek. Poizvedbe so zaradi tega zelo hitre, ponavadi kar v času $O(1)$, če so podatki dovolj majhni,

da jih lahko spravimo v pomnilnik.

Kot vidimo, podatki v takih bazah niso niti na noben način strukturirani niti ne obstajajo relacije med njimi. Uporabljajo se samo najpreprostejše poizvedbe. Agregati in podobno napredne funkcije ne obstajajo, temveč je treba vso morebitno obdelavo podatkov prenesti na aplikacijo.

Velika prednost baz ključ-vrednost je skaliranje, natančneje horizontalno skaliranje, ki je lahko zelo učinkovito in ga je preprosto izvesti. Poenostavljeno povedano: ključem vzamemo samo del njihove zgoščene vrednosti, nato pa jih glede na ta del razvrstimo po ostalih strežnikih.

Tipični predstavniki so Redis (podatki shranjeni samo v pomnilniku) [5], Memcached (podatki shranjeni samo v pomnilniku) [6] in Apache Cassandra (hibrid med SUPB tipa ključ-vrednost in stolpičnim SUPB) [32].

3.2 Stolpično usmerjeni SUPB

Stolpično usmerjene baze (angl. *column-oriented store*) so podobne bazam ključ-vrednost, vendar so narejene za bolj ekspresiven podatkovni model. V vsak stolpec lahko shranimo največ en (ali nič) par ključ-vrednost. Stolpce lahko združujemo skupaj v nadstolpce in tudi tem določimo nek ključ. Vsaka vrstica v bazi vsebuje več stolpcev ali nadstolpcev. Vse to je zelo podobno relacijskim bazam, vendar pa obstaja ena velika razlika. Stolpci in nadstolpci niso nujno identično organizirani po vrsticah, v relacijskih pa se moramo vedno prilagajati shemi. Prednost take hrambe pred relacijskim modelom je, da je baza časovno učinkovita tudi, ko ima zelo redke podatke (kar ne velja za relacijske podatkovne baze). Podatki v tovrstnih bazah so shranjeni v množici zgoščenih tabel, ki so v primeru nadstolpcev gnezdene ena v drugi. Tipična predstavnika sta Apache Cassandra [32] in Apache HBase [7].

3.3 Dokumentni SUPB

Jedro dokumentnih podatkovnih baz so, kot že ime pove, dokumenti. Dokumenti so lahko poljubnega tipa, npr. XML ali JSON. Vsakemu dokumentu pripada unikaten identifikator (id). S tega vidika so dokumentne baze podobne bazam ključ-vrednost. Dokumenti so vrednosti, id dokumenta pa je ključ. Poglavitna razlika je, da imajo dokumenti predpisano strukturo (npr. XML), ki lahko vsebuje različne elemente: podatkovne tipe, polja, sezname itd. To pomeni, je da je mogoče poizvedbe sedaj pisati tudi glede na vrednosti znotraj dokumenta. Če smo rekli, da pri bazah ključ-vrednosti to ni možno, pa pri dokumentnih podatkovnih bazah lahko uporabljamo vse napredne oblike poizvedb. Tipični predstavniki so MongoDB [8], Couchbase [9] in CouchDB [10].

3.4 Grafni SUPB

Grafne podatkovne baze oz. grafni SUPB sodijo med novejše produkte NoSQL. Grafne baze ponavadi uporabimo za shranjevanje močno povezanih podatkov. Podatki so predstavljeni v obliki grafa. V osnovi to pomeni, da imamo vozlišča, ki so med seboj povezana s povezavami. Vozliščem in povezavam pa lahko pripenjamo tudi poljubne atribute in lastnosti. Oblika grafa omogoča, da lahko, za razliko od relacijskih podatkovnih baz, učinkovito hranimo in interpretiramo relacije (povezave) med različnimi podatki.

Tipični predstavniki so Neo4j [11], OrientDB (grafno-dokumentni SUPB) [12], Titan [13] in FlockDB [33].

Poglavje 4

Grafni SUPB podrobno

Grafne podatkovne baze v osnovi omogočajo zelo podobne ali enake osnovne operacije, kot jih omogočajo relacijske (SQL) baze. To so: ustvari (angl. *create*), preberi (angl. *read*), izvedi (angl. *update*) in izbriši (angl. *delete*). Tako kot transakcijske, so tudi grafne baze narejene za delo v sistemih OLTP (angl. *online transaction processing*) [15]. Zelo pomembna lastnost grafnih baz je, da z večanjem podatkovne množice časovna učinkovitost poizvedb ne upade, temveč ostane konstantna. Grafne baze so narejene tako, da v množici vseh vozlišč vedno preiskujejo samo ožji obseg okoli začetnega vozlišča, večino zunanjih vozlišč pa pustijo nedotaknjenih [30].

Da za bazo lahko rečemo, da je grafna, mora na nek način implementirati t. i. *graf lastnosti* (angl. *property graph*). To pomeni, da morajo obstajati vozlišča, povezave in njim dodeljene lastnosti, omogočeno pa mora biti tudi preiskovanje grafa. Za dosego tega večina grafnih baz uporablja brez-indeksno naslavljanje podatkov [4]. Brez-indeksno naslavljanje, kot že ime pove, pomeni, da imajo podatki (vozlišča) na disku shranjene kazalce (lokacije) povezanih vozlišč. Nekatere grafne baze, kot je Titan, tega ne uporabljajo, slednji uporablja svojevrstno podatkovno strukturo, ki za vsako vozlišče hrani indekse sosednjih vozlišč [16]. Tudi zaradi tega smo se odločili, da uporabimo Neo4j, ki se še najbolj približa terminu nativna grafna baza. V tabeli 4 so prikazani trenutno najbolj popularni grafni SUPB in njihove

	Podjetje	**Jezik**	**Licenca**	**Poizvedovanje**
ArangoDB [19]	ArangoDB	C/C++	Apache2	Gremlin, AQL
AllegroGraph [20]	Franz inc.	Java	Pl., zastonjska	SPARQL, Prolog
Apache Giraph [21]	Apache	Java	Apache2	Java
FlockDB [33]	Twitter	Java	Apache2	Ruby
GraphBase [22]	FactNexus pty ltd	Java	Pl., zastonjska	Ruby
GraphPack [23]	Amit Portnoy	Java	Apache2	Java
HyperGraphDB [24]	kobrix.com	Java	LGPL	Java
InfiniteGraph [25]	Objectivity Inc.	Java, C++	Pl. zastonjska	Java, Gremlin
InfoGrid [26]	NetMash Inc.	Java	Plačljiva	Java
Neo4j [11]	Neo Technology Inc	Java	GPL/AGPL in pl.	Cypher, Gremlin
OrientDB [12]	NuvolaBase Ltd	Java	Apache2	SQL, Gremlin
Titan [13]	Thinkaurelius	Java	Apache2	Gremlin

Tabela 4.1: Primerjava nekaterih bolj popularnih grafnih baz.

lastnosti. Zanimivo je, da je večina baz odprtokodnih, nekatera podjetja pa ponujajo tudi plačljive različice. Pri SUPB smo opisali tudi načine za poizvedovanje in ugotovili, da jih veliko podpira poizvedbeni jezik Gremlin [17]. Čeprav še ni postal de facto standard za poizvedovanje, pa nekateri že primerjajo njegovo vlogo v grafnih bazah z vlogo jezika SQL v relacijskih SUPB [17].
Opisu baze Neo4j smo namenili celo poglavje 6, ostale grafne baze pa smo opisali v naslednjih podpoglavjih.

4.1 Prednosti grafnih baz

Predstavljamo nekaj razlogov, zaradi katerih se razvijalci odločajo za grafne baze. To so:

- Časovna učinkovitost. Uporabniki dandanes od spletnih aplikacij zahtevajo izjemno hitro odzivnost. Ena od lastnosti grafnih baz je, da se z večanjem količine podatkov časovna učinkovitost poizvedb ne poslabša. Obratno velja za relacijski SUPB: če se količina podatkov veča,

postajajo stiki (angl. *join*) vse bolj časovno zahtevni. V poglavju 9 smo podobno trditev o časovni učinkovitosti tudi preverili in izmerili na konkretnem primeru.

- Lažji in hitrejši razvoj. Objektno relacijska impedančna neusklajenost (angl. *object-relational impedance mismatch*) [18] je problematika, ki pesti relacijski SUPB, ko uporabljamo transakcijsko bazo v navezi z objektno usmerjenim jezikom. Objekti (v programskih jezikih) konceptualno ne spadajo v relacijski podatkovni model. Grafne baze so v tem pogledu konceptualno veliko bližje objektno usmerjenim jezikom, razvoj aplikacij pa je s tega vidika veliko lažji.

- Prilagodljivost na spremembe. Ko pride do sprememb v poslovni logiki ali realnih okoliščin, je te spremembe veliko lažje vključiti v grafno bazo. Oznaka, ki se uporablja za grafne baze uporablja je brez-shematska baza (angl. *schema-free*), kar v praksi pomeni, da baza nima omejitev, ki jih zapoveduje shema (npr. omejitev pri podatkovnih tipih). Dodajanje novih podatkov v tako bazo je veliko lažje kot pa v preteklosti, ko je bilo treba pri transakcijskih bazah spreminjati shemo.

4.2 OrientDB

OrientDB je trenutno tretji najbolj popularen grafni SUPB. Napisan je v Javi, koda pa je izdana pod odprtokodno licenco Apache 2.0. Zadnje velja samo za verzijo *community*, obstaja pa tudi plačljiva verzija, imenovana *enterprise*. OrientDB omogoča tri načine delovanja.

- Lokalni način. V tem načinu so podatki shranjeni na disku. OrientDB za ta način uporablja termin "plocal".

- Oddaljeni način. Hramba podatkov je enaka kot pri zgornji točki, le da do strežnika tokrat dostopamo oddaljeno. Dostop se vrši preko klicev REST. Način najdemo pod imenom "remote".

- Pomnilniški način. Podatki so shranjeni samo v pomnilniku. Gre za najhitrejši način, vendar zelo omejen glede količine podatkov, ki jih je možno shraniti. Termin, ki se uporablja za ta načina delovanja, je "memory".

Tako kot za Neo4j tudi za OrientDB obstaja način, da bazo vključimo v javansko aplikacijo. Iz aplikacije lahko tako dostopamo do baze v lokalnem načinu in se tako izognemo režijskim stroškom oddaljenega načina. Govorimo o t. i. vgrajenem načinu (angl. *embedded*).

SUPB OrientDB je bil zasnovan kot dokumentna baza, skozi čas pa je postala mešanica grafne in dokumentne baze. Vozlišča so še vedno shranjena kot dokumenti, v dokumentu, ki pripada vozlišču, pa je prisotna tudi informacija o lokaciji povezanih vozlišč. Prednost tega pristopa je, da podatki zavzamejo manj prostora, saj ni potrebnih nobenih zapisov o povezavah, slabost pa, da na ta način povezave ne morejo biti niti oboje smerne niti jim ne moremo dati lastnosti. Kot smo že omenili, naj bi bila lastnost grafnih SUPB ta, da so povezave entitete prvega reda. Tako jih npr. izpostavi tudi Neo4j, vendar pa OrientDB omogoča tudi to. Z vklopom določenih nastavitev lahko omogočimo "polne"povezave, ki jim lahko dodamo lastnosti in ki so lahko oboje smerne. Pri SUPB OrientDB govorimo o večmodelnem modelu (angl. *multi model*), kar pomeni, da bazo lahko uporabljamo kot grafno ali dokumentno ali pa mešanico obojega [38].

4.3 Titan

Tudi SUPB Titan je napisan v Javi in izdan pod odprtokodno licenco Apache 2.0. Avtorji navajajo, da gre za zelo razširljiv SUPB, ki je specializiran za grafe, distribuirane po več strežnikih [39]. Titan lahko deluje v dveh načinih. V bolj običajnem načinu, do oddaljene baze dostopamo preko klicev REST, pri vgrajenem načinu pa bazo vgradimo v javansko aplikacijo.
Titan za razliko od večine ostalih grafnih baz ne določa shranjevalnega pogona, pač pa lahko izbiramo med pogoni Apache Cassandra [32], Apache

HBase [7] in Oracle BerkeleyDB [27]. V splošnem je Titan zelo močno prepleten z različnimi rešitvami Apache, npr. za dostop do baze, poizvedovanje in druge funkcionalnosti se uporablja funkcionalnosti iz ekosistema Apache TinkerPop [28]. V našem raziskovanju grafnih baz smo v splošnem zelo velikokrat zasledili TinkerPop. Knjižnici, kot sta Gremlin [17] in Blueprints [29], sta postali najboljši približek standardizacije določenih stvari v svetu grafnih baz. Prvi je nekakšen SQL v grafnih bazah in ga privzeto poleg Titana podpira tudi OrientDB, drugi pa je nekakšen ODBC/JDBC grafnih baz. Za povezovanje preko Jave ga uporabljajo tako Titan kot tudi OrientDB in ArangoDB.

4.4 ArangoDB

ArangoDB spada v kategorijo več modelnih baz (tako kot OrientDB in Titan), saj združuje koncepte dokumentnih in grafnih baz. Baza ne uporablja shem. Tako lahko uporabniki podatke poljubno modelirajo kot: dokumente, pare ključ-vrednost ali grafe. ArangoDB, za razliko od preostalih opisanih baz, ne pozna vgrajenega načina, temveč samo oddaljeni (ali lokalni) dostop preko vmesnika REST. Za pisanje poizvedb se uporablja bazi lasten jezik, imenovan AQL (angl. *Arango query language*). Baza je napisana v jeziku C in je, podobno kot ostale omenjene, izdana pod odprtokodno licenco Apache 2.

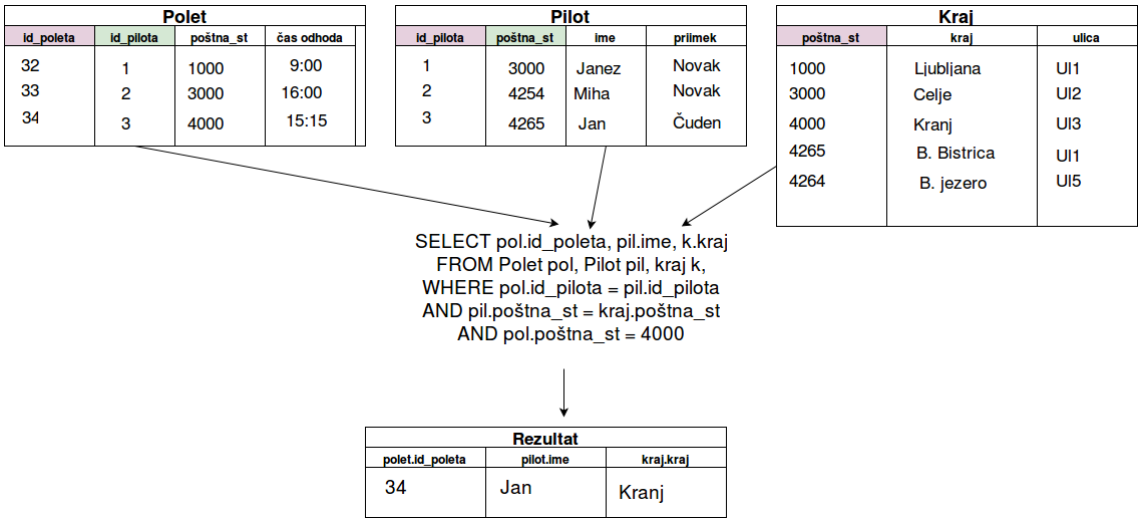
Poglavje 5

Relacijski SUPB

Relacijske podatkovne baze so bile in so še vedno najbolj popularna rešitev za shranjevanje vsakovrstnih podatkov. SUPB s trenutno največjim tržnim deležem je Oracle Database [37], po vrstnem redu od najbolj do najmanj popularnih pa mu sledi naslednjih 5 relacijskih SUPB:

- MySQL,
- Microsoft SQL server,
- PostgreSQL,
- DB2,
- Microsoft Access.

Pred trenutnim obdobjem socialnih omrežij in povezanih podatkov so ljudje večinoma shranjevali tabelarne, nepovezane podatke, kot so npr. podatki o poletih letal. Relacijski SUPB so za tovrstne podatke idealni. Če govorimo o primeru poletov, bi naredili tabelo za hranjenje informacije o poletu (npr.: id poleta , id pilota, id kraja, čas odhoda), posebno tabelo za pilote (npr.: id pilota, id kraja, ime, priimek), posebno tabelo za kraj (npr.: id kraja, kraj, poštna številka) itd. Da lahko iz teh tabel dobimo kompletne podatke o letu, moramo te nekako združiti. Za to moramo vpeljati primarne in tuje ključe.



Slika 5.1: Primer tabel in shranjenih podatkov v relacijskem SUPB.

Ključi, kot že ime pove, so ključni del relacijskih SUPB. Za primarne ključe moramo določiti stolpce, ki držijo unikatne vrednosti. V primeru poletov so to vsi stolpci, ki se začnejo z besedo id (id poleta, itd.). Tuja ključa pa sta stolpec z imenom "id pilota"v tabeli "polet", ter stolpec "id kraja"v tabeli "pilot".

Iz med zgoraj naštetih SUPB sta le MySQL in PostgreSQL odprtokodna. Tudi zaradi tega in zaradi splošne priljubljenosti, smo se odločili, da pri primerjavi grafnih in relacijskih SUPB uporabimo prav MySQL.

5.1 MySQL

MySQL je, kot smo videli v prejšnjem poglavju trenutno drugi najbolj uporabljani SUPB. Čeprav je izdan pod odprtokodno licenco GNU, si jo lasti podjetje Oracle, ki podjetjem ponuja tudi plačljive verzije. Popularnost je MySQL dosegel, kot osnovna komponenta zelo popularnega odprtokodnega razvojnega paketa LAMP. LAMP (in druge različice AMP za ostale operacijske sisteme) so popularizirale MySQL, kot bazo za razvoj raznovrstnih

spletnih strani, vendar pa bazo uporabljajo tudi večje korporacije, npr. Google in Twitter.

Ena iz med prednosti SUPB MySQL so različni shranjevalni mehanizmi (angl. *storage egine*), ki jih lahko uporabimo. Privzeti mehanizem se imenuje InnoDB, ki ga v praksi redko zamenjamo, vendar pa obstajajo specifični problemi, ki jih bolje rešujejo drugi pogoni, kot so npr. MyISAM (prvotni shranjevalni mehanizem), Memory (mehanizem, ki hrani vse podatke v pomnilniku za izjemno hitro obdelavo), mehanizem imenovan CSV, itd. Eden od ciljev te diplomske naloge je bil tudi, da preizkusimo rešitve, ki jih mogoče ponuja MySQL za boljše in hitrejše poizvedovanje po grafnih podatkih. Shranjevalni mehanizem, ki smo ga našli in ki naj bi bil prirejen za delo z grafi [35], se imenuje OQGGRAPH.

V času, ko je Sun (ki ga je kasneje prevzel Oracle) prevzel MySQL je skupnost (ker so nasprotovali prevzemu) naredila t.i. odcepitev (angl. *fork*) programske kode, ki so jo poimenovali MariaDB. MariaDB ohranja polno kompatibilnost z MySQL in se v splošnem razlikujeta zelo malo oz. za končne uporabnike nič. Uporabili smo MariaDB, da smo lahko namestili shranjevalni mehanizem OQGRAPH, ki ga MySQL ne podpira. Sintaksa poizvedb in časovna učinkovitost sta enaki, kot če bi uporabili MySQL.

5.1.1 Shranjevalni mehanizem OQGRAPH

Sprva smo zmotno domnevali, da gre pri OQGRAPH (angl. *Open query graph engine*) za shranjevalni mehanizem, ki je prirejen za hranjenje grafnih podatkov. Ugotovili smo, da gre v primeru OQGRAPH le za t.i. *računski mehanizem* (tako ga imenujejo avtorji). OQGRAPH je samo skupek funkcij, ki omogočajo lažje pisanje poizvedb nad grafi, hrambni pogon pa ostaja InnoDB. Elementi sintakse, ki jih uporablja, so zelo podobni MySQL. Novost je rezervirana beseda `LATCH`, s katero lahko določimo vrsto algoritma za preiskovanje (preiskovanje v širino ali globino, Dijkstrov algoritem itd.). Ker pa pri problemu prijatelj prijatelja v bistvu ne preiskujemo nič, temveč samo obiščemo vsa naslednja vozlišča iz začetnega (spustimo se samo za en nivo),

nam OQGraph ni bil v nobeno pomoč.

Tudi zato smo v diplomsko delo vključili še eno poizvedbo, ki se tiče problema iskanja najkrajše poti. Tovrstni problemi, kjer dejansko preiskujemo različne poti, pa so pisani na kožo mehanizmu OQGRAPH.

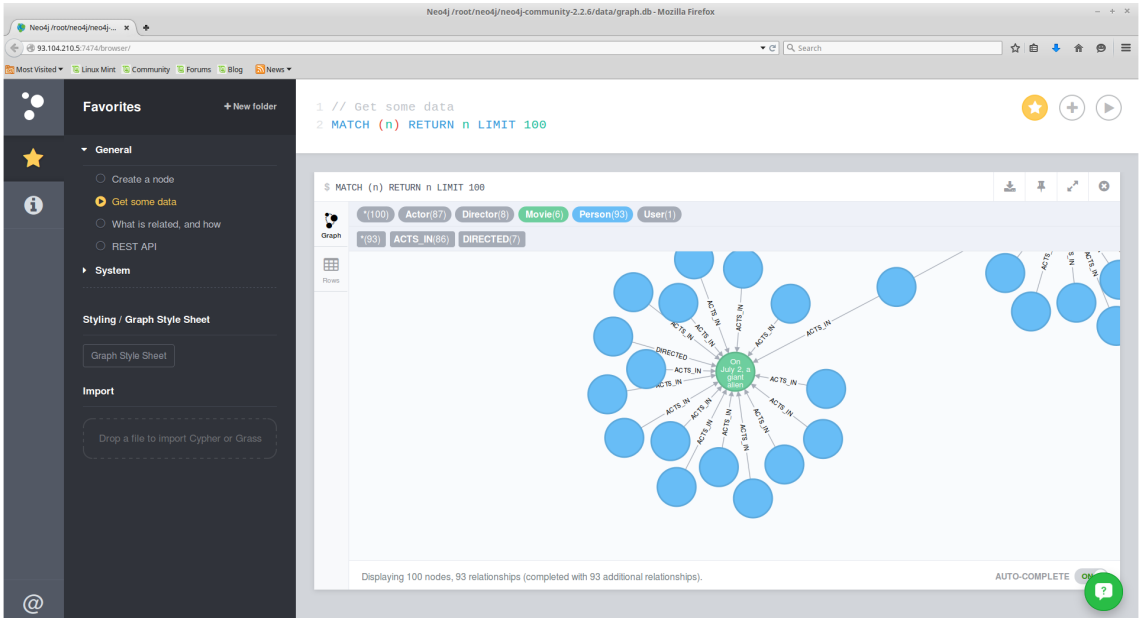
Poglavje 6

SUPB Neo4j

Neo4j je trenutno eden izmed bolj popularnih grafnih SUPB. Podjetje Neo Technology ponuja dve različici podatkovne baze: odprtokodno različico *community* pod licenco GPL3, ter zaprtokodno verzijo imenovano *enterprise*, ki je namenjena bolj komercialnim podjetjem. Pri izdelavi diplomskega dela smo uporabili odprtokodno verzijo, ki ponuja vse, kar smo potrebovali. Verzija *enterprise* ponuja nekaj dodatnih funkcionalnosti, kot sta npr. gručenje (angl. *clustering*) in izdelava vročih varnostnih kopij (angl. *hot backuping*, tj. izdelava varnostnih kopij pri delujoči bazi.). Med obstoječimi funkcionalnostmi med bazama ni razlike. V času nastajanja diplomskega dela ima zadnja verzija številko 2.3.

SUPB je napisan v Javi in kot tak deluje na vseh operacijskih sistemih, kjer domuje tudi Java. Za poizvedovanje lahko uporabljamo bazi lasten jezik imenovan Cypher. Cypher najlažje preizkusimo kar preko brskalniškega vmesnika, tako da (baza mora seveda biti zagnana) v poljuben brskalnik vtipkamo http://localhost:7474/browser/, če je strežnik nameščen na lokalnem računalniku, če ni, pa besedo *localhost* zamenjamo z naslovom IP oddaljene baze. Kar je mogoče malo nenavadno za nekoga, vajenega baz SQL, je, da ima podatkovna baza dva osnovna načina delovanja:

- Način REST. Je nekomu, ki prihaja iz vod SQL, bolj običajen način. Ime načina je kratica, ki označuje način komunikacije med klientom in

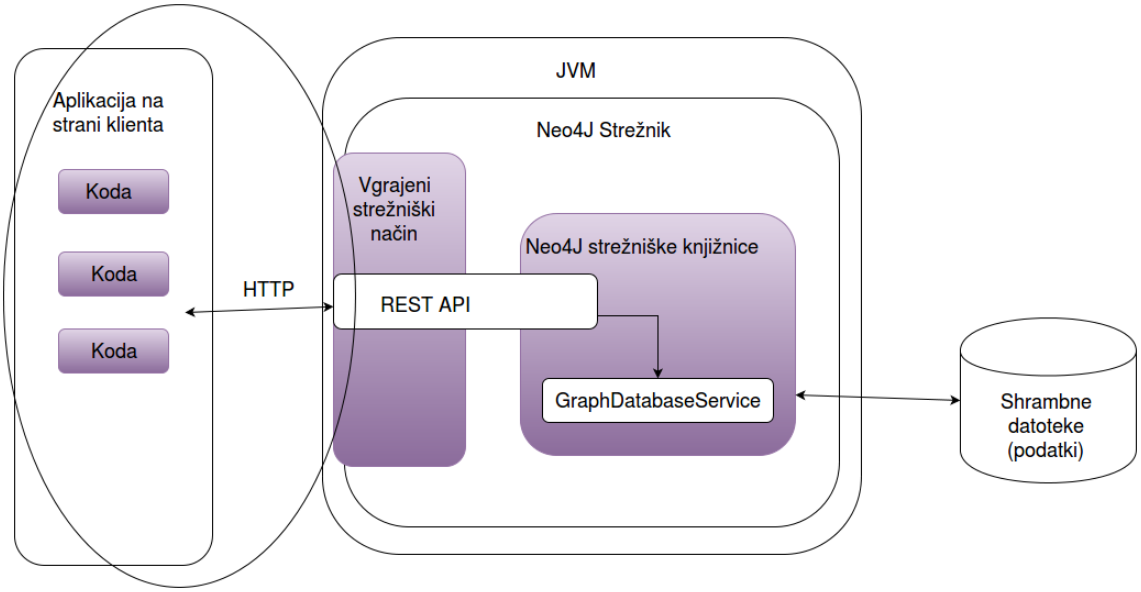


Slika 6.1: Brskalniški vmesnik, preko katerega lahko dostopamo do strežnika in izvajamo poljubne poizvedbe Cypher.

strežnikom - REST (angl. *Representational State Transfer*). Pri tem načinu gre za klasičen oddaljen dostop do baze, kjer moramo poizvedbe in ostale podatke nekako poslati preko mreže.

- Vgrajeni način (angl. *embedded mode*). Je način, pri katerem naša baza (strežnik) deluje znotraj naše javanske aplikacije. Iz naše primerjave v poglavju 9 ga bomo izpustili, saj ni smiselno primerjati dveh popolnoma drugačnih načinov uporabe baz. Vgrajeni način je zaradi svoje narave seveda tudi veliko hitrejši.

Oba načina, predvsem pa način REST, sta zelo pomembna za naše razumevanje delovanja baze Neo4j, in izvedbo ter interpretacijo rezultatov iz poglavij 9, in 9.1. Zaradi tega si ju podrobneje poglejmo v naslednjih dveh podpoglavjih.



Slika 6.2: Primer tipične uporabe strežniškega načina, kjer klienti do strežnika dostopajo preko običajnega načina REST.

6.1 Način REST

Če smo prej rekli, da gre za klasičen način dostopanja do baze, pa je malo bolj neobičajen način, ki ga baza za to uporablja. Podatki se pri temu načinu prenašajo po načinu REST, kar je z vidika relacijskih SUPB, ki uporabljajo gonilnik ODBC/JDBC, nenavadno. Je pa način REST kot tak čisto običajen v svetu sodobnih nerelacijskih podatkovnih baz. Poleg Neo4j ga uporabljata tudi OrientDB in Titan. Za REST pravimo, da ni protokol, temveč samo arhitekturni stil [40]. REST sicer ne določa temeljnega protokola za prenos podatkov, v praksi pa je to skoraj vedno protokol HTTP. REST je tako samo način, kako uporabljati metode HTTP, kot so GET, POST, PUT, DELETE idr.

Pri testiranju na praktičnem primeru v poglavju 9 se je izkazalo, da REST oz. natančneje protokol HTTP prinese s seboj kar nekaj režije (angl. *overhead*), posledično pa tudi počasnejše izvajanje poizvedb.

SUPB, kot je MySQL, za oddaljeni dostop ponavadi uporabljajo namenski gonilnik. Najbolj znana taka gonilnika sta ODBC in javanski JDBC, ki podatke posredujeta binarno preko vrat TCP. Kasneje smo ugotovili (in to tudi uporabili v naših primerjavah), da obstaja hitrejša implementacija načina REST, ki nosi ime JDBC [41]. Vendar tudi ta način ne komunicira direktno preko vrat, ampak gre samo za optimizirani način uporabe HTTP prenosa. Namenski gonilnik za način TCP/IP za Neo4j je bil v času pisanja diplomskega dela še vedno v razvoju.

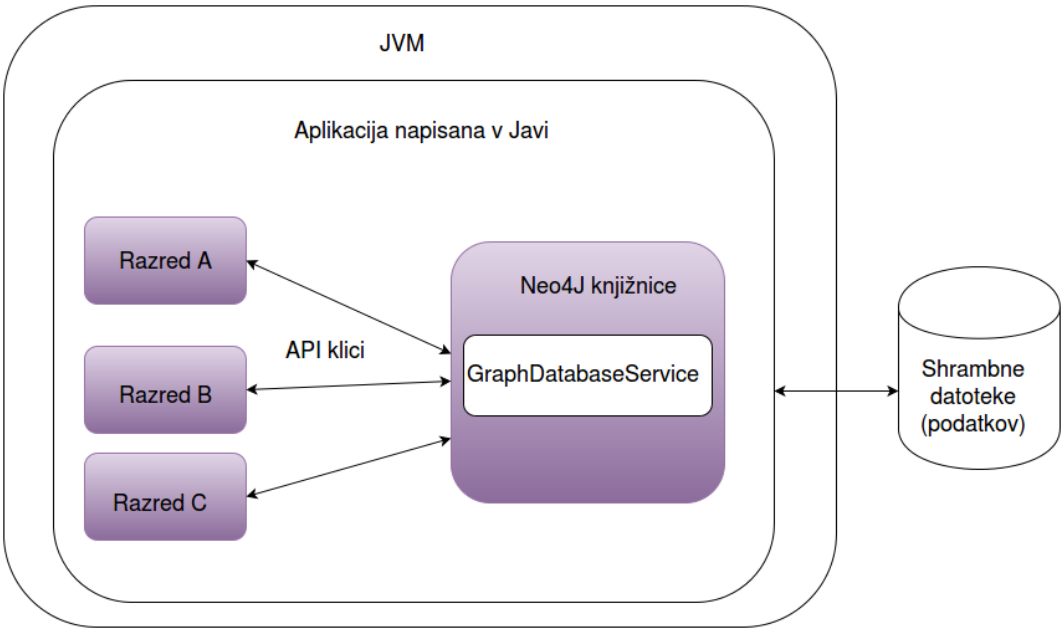
6.2 Vgrajeni način

Pri vgrajenem načinu gre pravzaprav za združitev aplikacije in strežnika. Ta način je trenutno omejen na uporabo Jave, kjer aplikacijska koda in strežniška tečeta na isti instanci JVM. Lahko si predstavljamo, koliko hitrejši je ta način od načina REST in koliko bolj omejen je, kar se tiče primerov uporabe, saj do baze lahko dostopamo samo iz ene in edine aplikacije, pri načinu REST pa je lahko klientov praktično neomejeno. Tudi nekateri drugi relacijski SUPB imajo možnost vgrajenega načina, npr. sqlite ali MySQL [42].

6.3 Poizvedbeni jezik Cypher

Cypher je poizvedbeni (tudi povpraševalni) jezik, ki je sestavni del SUPB Neo4j. Jezik bi lahko primerjali z jezikom SQL, ki je tako kot Cypher namenjen delu z bazami. Čeprav smo napisali, da je sestavni del baze, pa Neo4j omogoča tudi programatične sprehode po grafih [43]. Tako lahko npr. v Javi napišemo kodo, npr. zanko *for*, ki pregleduje vozlišča po globini ali širini. Za potrebe diplomskega dela smo za poizvedovanje uporabili jezik Cypher, saj smo se ga zelo hitro priučili. Je tudi veliko bolj jedernat od programatičnega načina.

Na prvi pogled zelo verjetno pri sintaksi jezika Cypher niti ne bi opazili njegove posebnosti. Sintaksa je namreč "grafična". Z "grafična"mislimo na



Slika 6.3: Primer uporabe vgrajenega načina, kjer so knjižnice Neo4j vgrajene v javansko aplikacijo.

to, da elementi sintakse, kot so oklepaji in zaklepaji, ter črtice in puščice, tvorijo grafične oblike, ki predstavljajo obliko vozlišč in obliko usmerjenih ali neusmerjenih povezav. Poglejmo si nekaj primerov poizvedb nad grafom prijateljstev. Spodaj predstavljena poizvedba vrne prijatelje osebe Blaž.

```
MATCH (jaz {name:"Blaž"})-[:JePrijatelj]->(prijatelji)
RETURN prijatelji
```

Poizvedbo razčlenimo na osnovne konstrukte in jo podrobneje opišimo:

- MATCH. Z besedo MATCH (slov. *enači*) ponavadi začnemo povpraševalno poizvedbo. Cypher sicer pozna tudi poizvedbe za kreiranje in brisanje vozlišč, katerih prvi besedi sta CREATE ali DELETE, ter druge vrste poizvedb, katerih namembnost pa nista iskanje in vračanje rezultatov. Pri naših primerih se bomo osredotočili samo na povpraševalne poizvedbe, saj smo tovrstne poizvedbe kasneje uporabili na dejanskem primeru. Besedi MATCH ponavadi sledi beseda WHERE, s katero določimo omejitve in predikate, ali pa ji sledi grafični vzorec vozlišč in povezav (kot v našem primeru). Beseda START, ki jo lahko zasledimo v nemalo primerih, pa je z novejšimi različicami jezika Cypher postala nepotrebna.

- `(jaz {name:"Blaž"})-[:JePrijatelj]->(prijatelji)`. Stavek, ki sledi v našem primeru, je vzorec dveh vozlišč, ki sta narisana z oklepaji, in usmerjene povezave med njima, ki je narisana kot črtica s puščico od enega vozlišča do drugega. Če zaradi lažje predstave iz vzorca odstranimo vse identifikatorje, dobimo vzorec:
  `( )-[:JePrijatelj]->( )`.
  `[:JePrijatelj]` je dejanski tip povezave v našem grafu, ki jo Neo4j uporabi za sprehod med vozlišči. Ime tipa povezave se mora vedno začeti z dvopičjem. S prej omenjenimi identifikatorji lahko vozlišča poljubno poimenujemo in si jih na ta način shranimo za poznejšo uporabo ali pa jih, kot v našem primeru, vrnemo kot rezultat poizvedbe. Vozlišča imajo po navadi shranjene tudi nekatere lastnosti, te pa lahko

v sintaksi navedemo znotraj zavitih oklepajev, kot pri našem primeru `(jaz {name:Blaž})`, kjer z {`name:Blaž`} povemo, da naj Cypher začne preiskovati v vozlišču, ki ima shranjeno vrednost `Blaž` pri lastnosti *name*. V tem primeru smo za risanje povezave uporabili usmerjeno povezavo, lahko pa bi uporabili tudi neusmerjeno, kot je npr.
`(jaz)-[:JePrijatelj]-(prijatelji)`.
Usmerjenost bi lahko obrnili:
`(jaz) <- [:JePrijatelj]-(prijatelji)`.
Vzorce vozlišč in povezav lahko poljubno podaljšujemo, ali pa zraven enega dodamo drugi vzorec, ki ju ločimo z vejico. Sintaksa torej dopušča bolj zapletene vzorce z več vozlišči in različnimi tipi povezav kot je npr. naslednji:
`(a)-[:JePrijatelj]->(b)-[:JeSodelavec]->(c)`.

- `(RETURN prijatelji)`. Da vrnemo najdene rezultate našega vzorca (`MATCH`), uporabimo besedo RETURN, ki ji dodamo želeno vozlišče, ki smo si ga prej shranili s poljubnim identifikatorjem.

Poglejmo si še en pogost primer poizvedbe, kjer želimo preiskovati samo določeno množico vseh vozlišč. To naredimo s predikatom, tako da uporabimo stavek WHERE. Vzemimo podoben primer prejšnjemu, le da tokrat hočemo poiskati samo ženske prijateljice osebe Blaž. To bi zapisali kot:

```
MATCH (jaz {name:"Blaž"})-[:JePrijatelj]->(prijateljice)
WHERE prijateljice.gender = 'female'
RETURN prijateljice
```

S stavkom WHERE omejimo preiskovanje vozlišč samo na tista, ki imajo pod lastnostjo `gender` zapisano `female`.

Cypher seveda ponuja veliko večji nabor stavkov, ki pa jih bomo samo omenili. To so:

- WITH. Pod-poizvedba znotraj MATCH poizvedbe.
- USING. Določimo indeks, ki ga želimo uporabiti.

- UNION. Združimo dve ali več poizvedb tako, da vrnejo eno množico rezultatov.
- ORDER BY. Uredi vrnjene rezultate.
- LIMIT. Omeji število vrnjenih vrstic.
- SKIP. Preskoči poljubno število vrstic in vrne vse naslednje vrstice.
- UNWIND. Z UNWIND razširimo kolekcijo v zaporedje vrstic.
- OPTIONAL MATCH. Določimo vzorec, ki ga želimo poiskati.

To poglavje je služilo kot kratek vpogled v zelo zanimiv poizvedbeni jezik. V poglavju 9 pa bomo videli tudi razliko med poizvedbama, napisanima v jezikih Cypher in SQL, oz. konkretneje v MySQL.

Poglavje 7

Namestitev in uporaba Neo4j

SUPB smo namestili na sistemu Linux, preizkusili pa smo tudi verzijo za sistem Windows, a kakšnih velikih razlik pri nameščanju ni bilo videti. Najbolj očitna je, da lahko na sistemu Windows za namestitev izberemo inštalacijsko datoteko, kjer nas skozi namestitev vodi program, na računalnikih z nameščenim operacijskim sistemov Linux pa si moramo prenesti arhivsko datoteko SUPB, ki jo na to razširimo v poljuben direktorij. Da strežnik poženemo, se prestavimo v ta direktorij in bazo poženemo z ukazom `./bin/Neo4j console` ali `./bin/Neo4j start`. Ko je SUPB zagnan, se nam v konzoli izpiše njegov naslov, ki je privzeto http://localhost:7474/.

Baza oz. podatki so privzeto shranjeni v datoteki `./data/graph.db`, lahko pa to tudi spremenimo, če odpremo konfiguracijsko datoteko `./conf/Neo4j-server.properties` in spremenimo vrstico, ki se nanaša na lokacijo podatkov. Ta datoteka in datoteka `Neo4j.properties` sta edini datoteki, ki jih potrebujemo za spreminjanje nastavitev Neo4j.

Graf oz. podatki, ki smo si jih izbrali, so že bili v formatu Neo4j in s končnico .db. Vse, kar je bilo treba narediti, je bilo v konfiguracijski datoteki spremeniti pot do podatkov. Namestitev baze oz. strežnika se ni izkazala za problematično. Več težav smo nato imeli z javanskim klientom.

7.1 Dostop do strežnika iz Jave

Da smo lahko oddaljeno dostopali do strežnika (način REST), je bilo treba vzpostaviti okolje z vsemi potrebnimi knjižnicami. Za razvoj smo uporabljali okolje Eclipse. Za klasičen način REST (ne tisti, ki uporablja t.i. način JDBC) je bilo treba uvoziti datoteko JAR, imenovano Neo4j-rest-graphdb, ki pa za delovanje potrebuje še vrsto dodatnih knjižnic. Uvažanje teh dodatnih knjižnic in iskanje pravih verzij se nista izkazala za dober način, saj nam ni uspelo dobiti delujočega projekta. Veliko boljši način je bil, da smo uporabili orodje Maven, katerega datoteko pom.xml, dobimo na naslovu https://github.com/Neo4j-contrib/java-rest-binding/blob/master/pom.xml. Pri temu načinu orodje Maven poskrbi za vse potrebne knjižnice, ki so naštete v datoteki pom.xml.

Za način JDBC, ki ni običajen način (uradna stran projekta Neo4j ga ne ponuja, kot možnosti), smo ironično porabili veliko manj časa. Zadeva je zelo preprosta. Z naslova https://github.com/Neo4j-contrib/Neo4j-jdbc prenesemo datoteko JAR, ki hrani tudi vse potrebne dodatne knjižnice. Datoteko JAR na to uvozimo v naš projekt na ustaljen način (odvisno od okolja, ki ga uporabljamo).

7.2 Uporaba

Uporaba knjižnic, katerih namestitev smo si pogledali v prejšnjem podpoglavju, ni težavna. Pri običajnem načinu REST, lahko postopek poizvedovanja razdelimo na dve točki:

- Povezava s strežnikom. S strežnikom se povežemo s pomočjo razreda `RestAPIFacade`, ki mu v konstruktor po vrsti navedemo argumente: naslov, uporabniško ime in geslo.

- Pošiljanje poizvedbenega niza. Za pošiljanje poizvedb uporabimo razred `QueryEngine`, ki mu v konstruktor podamo prej ustvarjeni objekt razreda `RestAPIFacade`. Na to uporabimo njegovo metodo `query`, ki

ji kot prvi argument podamo poljuben poizvedbeni niz. Kot drugi argument moramo navesti lastnost (angl. *property*) vozlišča, ki jo želimo prenesti (čez mrežo se prenese vedno samo ta del vozlišča in ne celo vozlišče, če seveda tako želimo). Primer kode, ki poskrbi za to se glasi: `QueryResult<Map<String,Object>> result = engine.query(query, map('coname', 0));`. Na tem primeru lažje pokažemo uporabo drugega argumenta, ki ga moramo podati v obliki podatkovne strukture map. Metoda `query` vrača rezultat tipa *QueryResult*, ki pa je v bistvu samo podatkovna struktura map, ki hrani rezultate vrnjene v obliki "ključ, vrednost".

Če to združimo, dobimo naslednje:

```
String query =
"MATCH (tom:Person {name: \"Tom Hanks\"})-[:ACTS_IN]
  ->(m)<-[:ACTS_IN]- coActors RETURN coActors.name as coName";

RestAPI graphDb =
    new RestAPIFacade("http://localhost:7474/","Neo4j","geslo");
QueryEngine engine=new RestCypherQueryEngine(graphDb);

QueryResult<Map<String,Object>> result =
engine.query(query, map("coName", 0));
```

7.2.1 Način JDBC

Način JDBC uporablja podoben postopek in enaka imena funkcij, kot jih uporablja gonilnik JDBC za SQL. Kot se spodobi, je treba pri klicanju funkcij uporabljati bloke kode za lovljenje izjem, ki pa jih bomo pri razlagi tega primera zaradi jedrnatosti kode izpustili.

Zopet bi lahko kodo razdelili na dva dela:

- Povezovanje s strežnikom. S strežnikom se povežemo s klicem metode razreda `DriverManager getConnection`, ki ji kot argumente podamo

naslov strežnika, uporabniško ime in geslo. Metoda vrača objekt tipa `Connection`.

- Pošiljanje poizvedbe in prejemanje rezultatov. Za izvedbo poizvedbe potrebujemo objekt tipa `Statement`, ki ga dobimo tako, da nad prej ustvarjenim objektom tipa `Connection`
  pokličemo metodo `createStatement`. Vse, kar še preostane, je da pokličemo metodo `Statement.executeQuery`, ki ji kot argument podamo poizvedbeni niz. Kot smo navajeni pri poizvedbah SQL, metoda vrača rezultat v obliki razreda `ResultSet`.

Če združimo, dobimo naslednje:

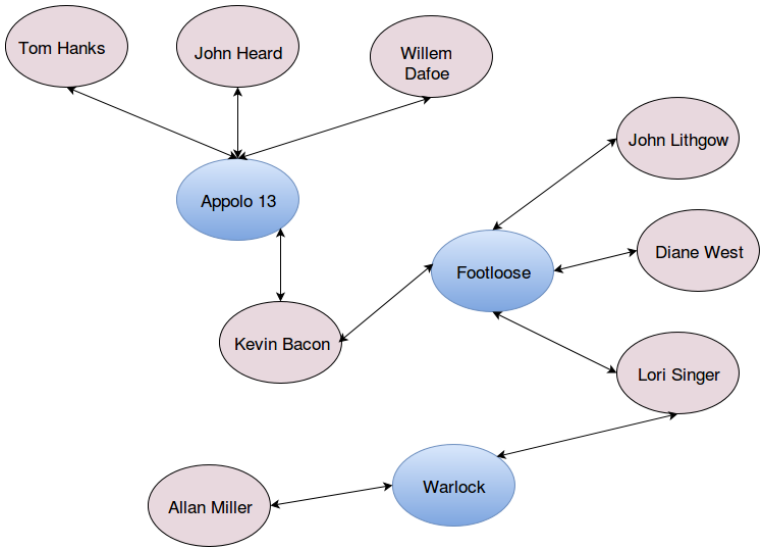
```
String query =
    "MATCH
    (tom:Person {name: \"Tom Hanks\"})-[:ACTS_IN]
    ->(m)<-[:ACTS_IN]- coActors RETURN coActors";

Connection con =
DriverManager.getConnection("http://localhost:7474/","Neo4j","pw");
Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(query);
```

Poglavje 8

Problem prijatelj prijatelja

Pri izdelavi diplomskega dela smo se odločili, da bomo različne SUPB testiral na problemu prijatelj prijatelja (angl. *friend of a friend*). Gre za konceptualni problem, ki se pojavi pri preiskovanju grafov prijateljstva. V prejšnjem poglavju smo jih poimenovali - socialni grafi. V praksi ta problem dobro rešuje Facebook, ko ponuja, da bi dodali nove prijatelje, ki jih morebiti poznamo. V resnici ni nujno, da jih poznamo, ti predlogi so samo prijatelji naših prijateljev. Prijatelj prijatelja je le eden izmed množice primerov problemov, ki jih imajo velika podjetja, kot so Facebook in Twitter, ki imajo opravka z več milijoni vozlišč velikimi grafi prijateljstva. Za to sta na primer Twitter in Facebook razvila lastne in posebej za te primere optimizirane SUPB (npr. Cassandra [32], ki jo je do pred kratkim ogromno uporabljal Facebook ali FlockDB [33], ki so jo razvili pri Twitterju za potrebe iskanja presekov med sledilci).

Problematike prijatelj prijatelja smo se v diplomskem delu lotili s stališča manjših podjetij, ki imajo na eni strani na voljo rešitev z uporabo tabel in stikov v klasičnem relacijskem SUPB, npr. MySQL, na drugi strani pa uporabo grafne baze, kot so Neo4j, TitanDB, OrientDB ali MySQL z vtičnikom OQGRAPH.



Slika 8.1: Poenostavljen primer naših podatkov. Na sliki lahko vidimo tri nivoje prijateljstev Toma Hanksa.

8.1 Graf igralcev in filmov

Na sliki 8.1 je prikazan poenostavljen primer naše podatkovne množice (pravi podatki vsebujejo tudi povezave tipa "je režiral"in podrobnosti o igralcih in filmih). Z elipsami smo predstavili igralce, puščice pa pomenijo, da je igralec igral v filmu. Na sliki vidimo več nivojev t. i. prijateljstev, natančneje vidimo tri nivoje. Prvi nivo pomeni, da poiščemo vse soigralce, s katerimi je kadar koli igral Tom Hanks. To so na sliki njegovi soigralci v filmu Apollo 13: John Heard, Williem Dafoe in Kevin Bacon. Če sledimo povezavi od igralca Kevina Bacona do filma Footloose, dobimo drugi nivo prijateljstva. To so vsi soigralci igralcev, s katerimi je kadarkoli igral Tom Hanks. Tretji nivo na sliki bi tako bil igralec Allan Miller.

V naslednjem razdelku bomo predstavili, kako bi bil tak graf implementiran v relacijskem SUPB (npr. MySQL) in kako bi izgledalo poizvedovanje po prijateljih od prijateljev.

	Ana	Bojan	Cene
Ana	1	1	1
Bojan	1	1	0
Cene	1	0	1

Tabela 8.1: Matrika sosednosti.

8.2 Rešitev v relacijskih SUPB

Za predstavitev grafa s podatkovnimi strukturami imamo dve možnosti: kot seznam sosedov (angl. *adjaceny list*) ali kot matriko sosednosti (angl. *adjacency matrix*). Seznam sosedov je definiran kot polje (ali seznam) vseh vozlišč, vsako izmed teh vozlišč pa ima shranjen povezan seznam (angl. *linked list*) vseh vozlišč, do katerih vodijo povezave iz navedenega vozlišča. Primer, kako bi izgledal seznam sosedov za sliko 8.1 (primer je nepopoln, ostala imena bi dodali po istem postopku), je naslednji:
$Ana|Ana \rightarrow Bojan \rightarrow Cene$
$Bojan|Bojan \rightarrow Ana \rightarrow Danijel$
$Evgen|Evgen \rightarrow Frenk \rightarrow Gaja \rightarrow Helena$
Matrika sosednosti ima po stolpcih in vrsticah navedena vsa vozlišča. Z vrednostjo 1 označimo, če med dvema vozliščema obstaja povezava, z vrednostjo 0 pa, če povezave ni. Primer matrike sosednosti je 8.2.

Matrika sosednosti trpi za dvema problemoma:

- Podatkovne baze imajo omejitev na največje število stolpcev v tabeli. Pri MySQL je teoretično lahko največ 4096 stolpcev, praktično pa še manj.

- Če hočemo dodati ali odstraniti vozlišče, moramo to storiti z ukazom tipa DDL, natančneje z ukazom *alter*. Ukazi DDL spreminjajo strukturo baze in so zelo počasni v primerjavi z ukazi tipa DML (npr. *insert*).

Zaradi naštetih omejitev se matrik sosednosti po navadi ne uporablja. V praksi se po navadi graf predstavi z dvema ločenima tabelama: tabelo vozlišč

in tabelo povezav.

```
tabela Vozlišča(PRIMARY KEY INTEGER vozlišče_id, VARCHAR(20) ime)

tabela Povezave(INTEGER a, INTEGER b,
FOREIGN KEY (a) REFERENCES Vozlišča(vozlišče_id),
FOREIGN KEY (b) REFERENCES Vozlišča(vozlišče_id)
)
```

Tabela *Vozlišča* hrani id-je vozlišč, tabela *Povezave* pa te id-je uporablja kot tuje ključe (angl. *foreign key*) začetnega in končnega vozlišča te povezave. Da poizvedbo pohitrimo moramo dodati indeks nad stolpce, ki se uporabljajo v stavku WHERE, tj. ime igralca. Dodajanju indeksa se lahko izognemo tako, da v pogoju WHERE stavka uporabimo id-je igralcev in ne imen.

8.3 Opis poizvedb

V diplomskem delu smo opisali in navedli samo poizvedbe na prvem nivoju, poizvedbe na nivoju 2 in 3 pa so priložene v dodatku. Poizvedba v jeziku Cypher je zelo preprosta in lahko razumljiva, kar pa ne moremo trditi za poizvedbo v jeziku MySQL. Celotna Cypher poizvedba se glasi:

```
MATCH (Tom:Person{name:"Tom Hanks"})-[:ACTS_IN]->
    (m)<-[:ACTS_IN]-coActors
RETURN coActors
```

Celotno poizvedbo lahko razčlenimo na dva dela. Najprej moramo dobiti vse filme v katerih je igral Tom Hanks. Za to začnemo poizvedovati v vozlišču "Tom Hanks"in pregledujemo vse povezave z imenom "igra v". To je narejeno s sledečim delom poizvedbe:

```
Tom:Person{name:"Tom Hanks"})-[:ACTS_IN]->(m)
```

Iz drugega konca sestavimo zrcalno poizvedbo prejšnji. Poizvedbo beremo od desne proti levi:

```
(m)<-[:ACTS_IN]- coActors
```

Poizvedba preiskuje igralce (označimo s sinonimom "(coActors)"), ki so igrali v prej dobljenih filmih (sinonim (m)). Igralce skupaj z vsemi pripadajočimi lastnostmi vrnemo s stavkom *return*.
Za dosego istega cilja je treba v MySQL vložiti nekoliko več truda. Celotna poizvedba SQL se glasi:

```
SELECT person.id, person.name, person.birthday,
person.birthplace, person.profileImageUrl, person.biography,
person.version, person.lastModified FROM person
JOIN relation ON person_id = person.id WHERE movie_id IN
(SELECT movie_id FROM relation WHERE person_id =
(SELECT id FROM person
WHERE name = 'Tom Hanks'))
AND person.id !=
(SELECT id FROM person WHERE name = 'Tom Hanks')
AND type = 'acts_in'
```

8.3.1 Poizvedbe v MySQL

Kot smo že omenili, je treba za prenos grafa v relacijski SUPB ustvariti vsaj dve tabeli. Mi smo ustvarili tri. Dve od tabel sta namenjeni vozliščem: eno za vozlišča s tipom igralec (poimenovano *person*) in drugo za vozlišča tipa film (poimenovano *movie*). Tretja tabela *relation* hrani povezave med vozlišči tako, da vsebuje stolpec z id-ji igralcev, ter stolpec z id-jem filma, kjer je igral. Celotna relacijska shema tabel je predstavljena na sliki 8.2. Poizvedba deluje tako, da z zadnjim stavkom *SELECT* dobimo id igralca, čigar soigralce bomo iskali. Nato na osnovi tega id-ja s stavkom *SELECT* iz tabele *relation* poiščemo njegove soigralce.

```
MariaDB [neo4jtest]> describe neo4jtest.movie;
+--------------+---------------+------+-----+---------+-------+
| Field        | Type          | Null | Key | Default | Extra |
+--------------+---------------+------+-----+---------+-------+
| id           | int(11)       | NO   | PRI | NULL    |       |
| title        | varchar(256)  | NO   | MUL | NULL    |       |
| tagline      | varchar(2048) | YES  |     | NULL    |       |
| description  | text          | YES  |     | NULL    |       |
| imageUrl     | varchar(2048) | YES  |     | NULL    |       |
| genre        | varchar(45)   | YES  |     | NULL    |       |
| studio       | varchar(128)  | YES  |     | NULL    |       |
| trailer      | varchar(2048) | YES  |     | NULL    |       |
| imdbId       | varchar(45)   | YES  |     | NULL    |       |
| version      | int(11)       | YES  |     | NULL    |       |
| homepage     | varchar(2048) | YES  |     | NULL    |       |
| runtime      | tinyint(5)    | YES  |     | NULL    |       |
| releaseDate  | timestamp     | YES  |     | NULL    |       |
| language     | varchar(3)    | YES  |     | NULL    |       |
| lastModified | timestamp     | YES  |     | NULL    |       |
+--------------+---------------+------+-----+---------+-------+
15 rows in set (0.00 sec)

MariaDB [neo4jtest]> describe neo4jtest.person;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| id              | int(11)       | NO   | PRI | 0       |       |
| name            | varchar(256)  | NO   | MUL | NULL    |       |
| birthday        | timestamp     | YES  |     | NULL    |       |
| birthplace      | varchar(2048) | YES  |     | NULL    |       |
| profileImageUrl | varchar(2048) | YES  |     | NULL    |       |
| biography       | text          | YES  |     | NULL    |       |
| version         | int(11)       | YES  |     | NULL    |       |
| lastModified    | timestamp     | YES  |     | NULL    |       |
+-----------------+---------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

MariaDB [neo4jtest]> describe neo4jtest.relation;
+-----------+----------------------------+------+-----+---------+-------+
| Field     | Type                       | Null | Key | Default | Extra |
+-----------+----------------------------+------+-----+---------+-------+
| person_id | int(11)                    | NO   | MUL | NULL    |       |
| movie_id  | varchar(45)                | NO   |     | NULL    |       |
| name      | varchar(256)               | YES  |     | NULL    |       |
| type      | enum('acts_in','directed') | YES  |     | NULL    |       |
+-----------+----------------------------+------+-----+---------+-------+
```

Slika 8.2: Relacijska shema tabel *movie*, *person* in *relation*.

8.3.2 Poizvedba v OrientDB

OrientDB smo dodali naknadno, zato bo poizvedba opisana bolj jedernato. Jezik je sintaktično zelo podoben jeziku SQL. Od njega se loči predvsem po nekaj ključnih besedah (funkcijah), kot so npr: *out* (odpre povezavo usmerjeno na ven), *in* (odpre povezavo usmerjeno na notri), *both* (odpre povezavo ne glede na usmerjenost),*expand* (razširi dokument), idr..

```
SELECT * FROM (SELECT expand(out('ACTS_IN').in('ACTS_IN'))
  FROM Person WHERE name = 'Tom Hanks')
WHERE name <> 'Tom Hanks';
```

Opisana poizvedba ni prva poizvedba, ki smo jo sestavili. Prvotna poizvedba je bila za več, kot 100 krat počasnejša. Ugotovili smo, da je zelo pomembno na kakšen način in v kakšnem zaporedju uporabljamo odpiranje povezav (funkciji *in*, *out*).

Poglavje 9

Primerjava časovne učinkovitosti poizvedb

Da smo dobili čim bolj merodajne rezultate, smo strežnika Neo4j in MySQL inštalirali na oddaljenem računalniku, aplikacijo pa smo poganjali na domačem računalniku. Za podatke (graf), nad katerimi smo poganjali poizvedbe smo si izbrali bazo filmov, igralcev in režiserjev, ki smo jo prenesli iz naslova [44]. Graf vsebuje 63042 vozlišč, od tega 50179 oseb, ostalo so filmi, in 213302 povezav med vsemi vozlišči. Povezave so lahko različnega tipa: "režiral", "igral v", "ocenil"in "prijatelj od". V našem primeru, ko bomo iskali soigralce, se bomo po grafu sprehajali samo po povezavah "igral v".

Za naš graf lahko rečemo, da je zelo povezan (govorimo o gostem grafu). To pomeni, da če hočemo poiskati vse soigralce soigralcev Toma Hanksa (nivo 2), dobimo vrnjenih 29606 rezultatov, kar predstavlja skoraj 2/3 vseh oseb. Da bi lahko primerjali tudi primer, ko poizvedba ne vrne tako veliko rezultatov, smo poleg soigralcev Toma Hanksa iskali tudi soigralce manj znanega Douga Spinuzze.

	Doug1	Tom1	Doug2	Tom2	Doug3	Tom3	NajkrajšaP
Neo4j-REST	337	391	460	1038	1356	20134	461
Neo4j-JDBC	157	377	877	912	1217	18566	158
MySQL	768	1070	1055	4466	ni končal	ni končal	225
OrientDB	80	119	5460	9670	9670	29243	110

Tabela 9.1: Rezultati meritev za oba igralca na vseh treh nivojih (številka ob imenu) in problem iskanja najkrajše poti (od Toma Hanksa do Mela Gibsona) v milisekundah.

9.1 Rezultati meritev

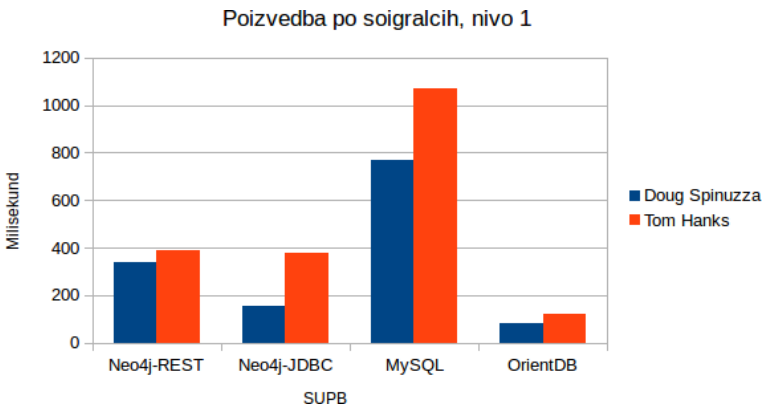
Osnovna zamisel diplomskega dela je bila, da naredimo primerjavo dveh SUPB: Neo4j in MySQL, pri čemer smo pri Neo4j uporabili dva načina (REST in JDBC). Za vsak način smo izvedli 6 vrst poizvedb: na vseh treh nivojih smo iskali soigralce Toma Hanksa in soigralce Douga Spinuzze.

Nato smo se odločili, da raziskovalno delo še razširimo. Napisali smo poizvedbe še v jeziku SUPB OrientDB in dodali novo poizvedbo, nad enako podatkovno množico, kot pri problemu iskanja soigralcev. Pri novi poizvedbi gre za grafni problem iskanja najkrajše poti med dvemi vozlišči, konkretneje, iskali smo najkrajšo pot od Toma Hanksa do Mela Gibsona (nikoli nista igrala skupaj).

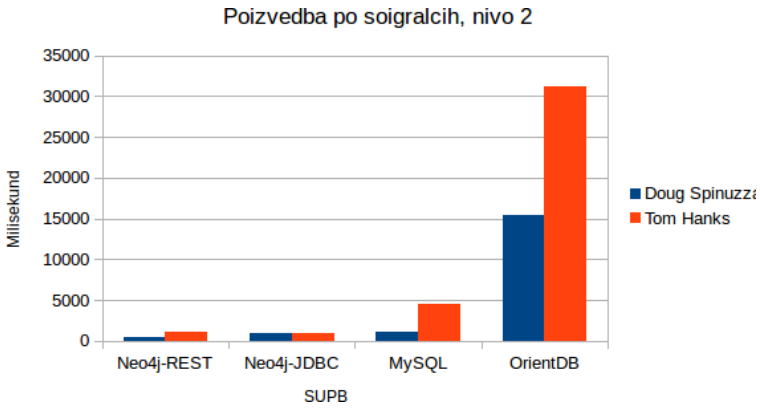
Vsako poizvedbo smo pognali večkrat, da smo dobili povprečne čase, ki jih navajamo na sliki 9.1 in v tabelah 9.1 in 9.3.

9.1.1 Rezultati meritev problema iskanja najkrajše poti

Pri snovanju diplomskega dela smo hoteli raziskati in uporabiti tudi dodatne shranjevalne mehanizme za relacijske SUPB. Dodatni shranjevalni mehanizem, ki naj bi bil prirejen za delo z grafi v MySQL oz. MariaDB se imenuje OQGRAPH. Ker je problem iskanja najkrajše poti zelo težko realizirati v običajnem MySQL (tovrstni problem se ponavadi ne rešuje v SQL ampak se ga preloži na programski jezik oz. aplikacijo), smo ga pri testiranju in



Slika 9.1: Rezultati iskanja soigralcev na prvem nivoju.



Slika 9.2: Rezultati iskanja soigralcev na nivoju 2.

meritvah nadomestili z mehanizmom OQGRAPH.

Poizvedbe, ki smo jih uporabili v OrientDB in Neo4j, smo vključili v dodatek A. Obe poizvedbi sta bili relativno preprosti za napisati, saj se uporabljata že vgrajeno funkcijo iskanja najkrajše poti (funkcija *shortestPath*). Nekaj več truda je bilo potrebno vložiti, da smo lahko uporabili OQGRAPH.

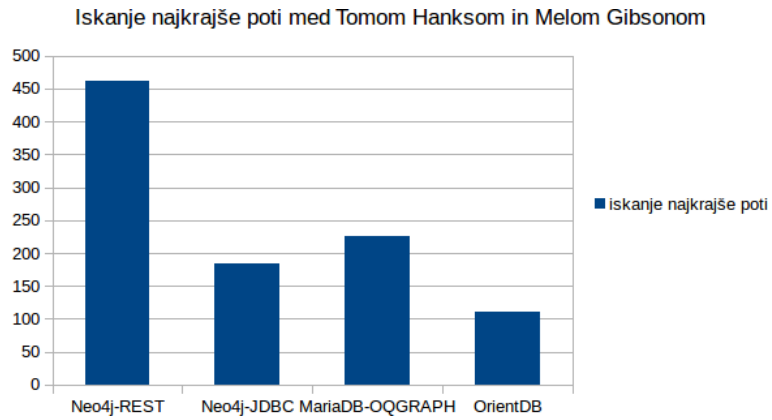
Iskanje najkrajše poti z OQGRAPH

Podobno kot pri Neo4j in OrientDB je poiskati najkrajšo pot med dvema vozliščema zelo enostavno, saj je iskanje najkrajše poti (z poljubnim algoritmom) osnovna namembnost mehanizma OQGRAPH. Zahteva, ki jo postavi OQGRAPH je, da moramo ustvariti novo tabelo povezav (ki jo moramo poimenovati po osnovni tabeli povezav in pa ji na konec še dodati končnico *_oq*). V našem primeru je to tabela *acts_in_oq*. Ta tabela hrani samo dva stolpca: *origid* in *destid*. OQGRAPH nato sam na novo zgradi in napolni tabelo poimenovano *acts_in_oqgraph*. Tudi ta tabela ima oba stolpca povezav (*origid* in *destid*), dodanih pa je še nekaj novih stolpcev: *latch, weight, seq, linkid*. Poizvedba, ki smo jo uporabili:

```
SELECT * from acts_in_oqgraph WHERE latch = 1 AND origid = 1 AND
destid= 2461
```

V poizvedbo moramo samo še vključiti začetno vozlišče (origid) in pa končno (imenovano destid), poimenovanji vozlišč pa morata biti obvezno ravno taki. Poizvedbi dodamo še rezervirano besedo *latch* in številko željenega preiskovalnega algoritma (privzeto je to 1, kar pomeni Dijkstrin algoritem). Za vse ostalo poskrbi mehanizem OQGRAPH.

Vendar pa se je tukaj zadeva malenkostno zapletla. Igralci v naši bazi seveda niso povezani nesporedno, temveč tvorijo enosmerne povezave s filmi v katerih so igrali. Da algoritmi (ki jih uporablja OQGRAPH) lahko opravijo sprehod po takem grafu, smo morali povezave še obrniti. To naredimo tako, da za vsako povezavo dodamo novo vrstico, pri čemer zamenjamo vrednosti stolpcev *origid* in *destid*.



Slika 9.3: Rezultati iskanja najkrajše poti.

9.2 Kaj to pomeni

Nesporni zmagovalci meritev so grafni SUPB. Težko pa bomo določili konkretnega zmagovalca pri grafnih SUPB. MySQL se je v vseh primerjavah uvstil najslabše. Če se osredotočimo na najbolj enostaven problem, tj. poizvedba po soigralcih Douga Spinuzze ugotovimo, da je MySQL dvakrat počasnejši od Neo4j-REST in štirikrat od Neo4j-JDBC. Na nivoju 1 opazimo, da je nesporno najhitrejši OrientDB. Na nivoju 2 pri poizvedbi *Tom Hanks*, so časi SUPB MySQL slabši od Neo4j že za faktor 4. Razlika pa je največja na nivoju tri, kjer rezultatov v primeru SUPB MySQL nismo dobili. Poizvedbe smo po več kot petih urah izvajanja prekinili. Pri bazi MySQL lahko razloge iščemo v ogromnih stikih, ki jih ustvarimo v poizvedbi. Poizvedba, ki se izvrši, vsebuje dva leva stika nad tabelama oseb in povezav. To v najslabšem primeru pomeni, da se eden iz med stikov izvede nad polnima tabelama. Dobimo kartezični produkt s 4.751.951.300 vrsticami (94.700 vrstic povezav $*$ 50.179 vrstic oseb, kar pa je očitno prevelik zalogaj).

Zelo dobro pa se MySQL (s shranjevalnim mehanizmom OQGRAPH) izkaže pri problemu iskanja najkrajše poti. Čeprav so časi meritev za približno polovico slabši od OrientDB in za 18 odstotkov od Neo4j-JDBC, lahko rečemo, da so razlike majhne. Iz rezultatov je razvidno, da je OQGRAPH

za polovico hitrejši od Neo4j-REST.

9.2.1 Problematika meritev pri OrientDB

Rezultati meritev kažejo na močno neskladje. Kot smo že omenili, se pri poizvedbi po soigralcih na nivoju 1 najbolje odreže OrientDB, prav tako je najhitrejši tudi pri iskanju najkrajše poti. Nasprotno pa se izkaže kot absolutno najpočasnejši na nivoju 2 pri iskanju soigralcev. Podobno počasen je v primerjavi z Neo4j tudi na nivoju 3. Razlog za to je da nista poizvedbi napisani optimalno. OrientDB uporablja poizvedbeni jezik, ki je sintaktično zelo podoben jeziku SQL. Mogoče je napisati veliko različnih poizvedb, ki sicer vračajo isti rezultat, vendar pa so posledično nekatere počasnejše od drugih. Krivdo lahko vsaj delno prevalimo na OrientDB, ki poizvedb ne optimizira najbolje, oz. na poizvedbeni jezik, ki ni tako nedvoumen, kot je npr. jezik Cypher.

9.3 Primerjava podatkovnih baz glede na ostale kriterije

V diplomskem delu smo podatkovne baze do sedaj primerjali samo glede na kriterij hitrosti. V celotnem obdobju izdelave diplomskega dela pa smo, z uporabo in predvsem skozi različne probleme, dodobra spoznali tudi druge plati. Podatkovne baze bomo primerjali in ocenili še glede na kriterija: udobnosti poizvedovanja in funkcionalnosti.

9.3.1 Kriterij udobnosti poizvedovanja

Tekom dela na diplomskem delu se nam je zelo priljubil poizvedovalni jezik Cypher, nasprotno pa lahko rečemo, da ne maramo poizvedovalnega jezika, ki ga uporablja OrientDB. Za to imamo dva razloga. Prvi razlog je, da je bilo poizvedbe težko napisati in razumeti. Če pri enostavnih poizvedbah še gre, pa so bile bolj kompleksne poizvedbe veliko težje razumljive. Drugi razlog pa

je ta, da je isto poizvedbo (tako ki vrača enake rezultate) mogoče sestaviti na veliko načinov. Kako poizvedbo sestavimo, pa bo lahko tudi zelo vplivalo na čas izvajanja le te. Najlažje to pokažemo na primeru naše poizvedbe po soigralcih (nivo 1).

```
SELECT name as name FROM
  (SELECT expand(in('ACTS_IN')) FROM Movie
  WHERE in('ACTS_IN').name in 'Tom Hanks')
WHERE name <> 'Tom Hanks';
```

Zgoraj zapisana poizvedba, je naša prva poizvedba, ki smo jo sestavili sami. Čas izvajanja: 10.12 s.
Sledeča poizvedba pa je izboljšana različica prve. Obe poizvedbi vračata iste rezultate. Čas izvajanja: 0.20 s.

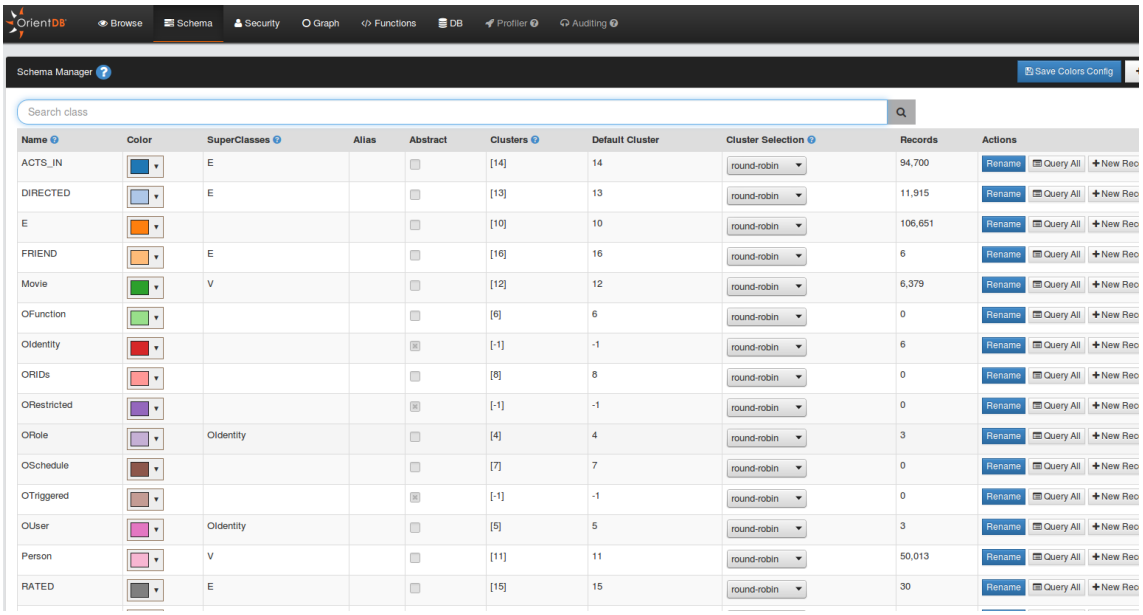
```
SELECT name as name FROM
  (SELECT expand(out('ACTS_IN').in('ACTS_IN'))
  FROM Person WHERE name = 'Tom Hanks')
WHERE name <> 'Tom Hanks';
```

Spodaj je napisana poizvedba v jeziku Cypher, ki bi jo težko napisali kako drugače.

```
MATCH
(tom:Person {name: "Tom Hanks"})-[:ACTS_IN]->
  (m) <-[:ACTS_IN]- coActors
RETURN coActors.name;
```

9.3.2 Kriterij funkcionalnosti

Pri izdelavi diplomske naloge smo pri vseh SUPB uporabljali samo čisto osnovne funkcionalnosti. Funkcionalnosti, kot sta npr: gručenje in podpora

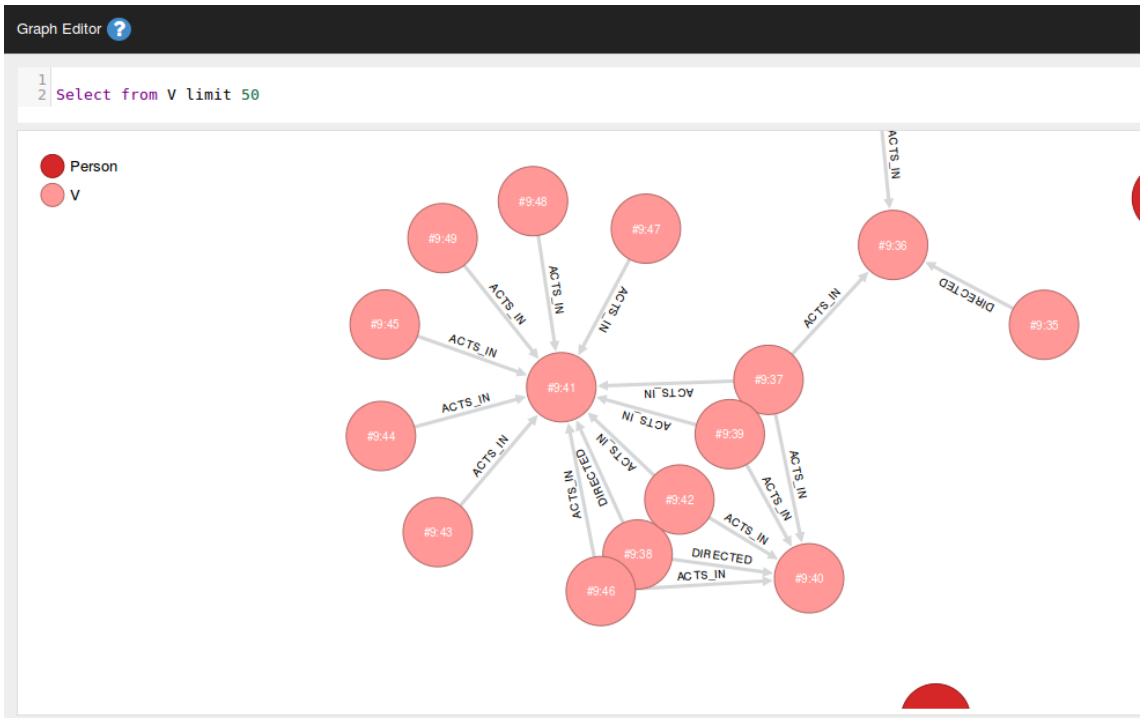


Slika 9.4: Pregled baze (sheme) v vmesniku za OrientDB.

večnitnosti bomo pri pregledu izpustili, osredotočili pa se bomo na tisto, ki smo jo tudi sami uporabljali. Zagotovo najpogosteje uporabljan je bil spletni vmesnik, ki smo ga večino časa uporabljali, da smo dostopali do baze in izvajali poizvedbe. Oba grafna SUPB ponujata tovrstni vmesnik, ne pa tudi MySQL (obstajajo pa zunanji spletni vmesniki, npr: phpMyAdmin).

9.3.3 Spletni vmesnik

Spletni vmesnik za Neo4j ponuja manj funkcionalnosti, kot vmesnik za OrientDB. Stvar, ki smo jo najbolj pogrešali je osnovni pregled baze (sheme), kot ga nudi vmesnik za OrientDB na sliki 9.5. Tudi grafični pregled baze v grafni obliki je bolje izveden v spletnem vmesniku za OrientDB (slika **??**), kjer vozlišča in povezave lahko urejamo, dodajamo nova, ali pa zbrišemo že obstoječe. Pogled v graf lahko poljubno približamo ali oddaljimo, vozlišča pa lahko tudi prostorsko preuredimo. Spletni vmesnik za Neo4j ne nudi ničesar od prej naštetega, omogoča samo enostavno pregledovanje vozlišč in povezav.



Slika 9.5: Grafični pogled v vmesniku za OrientDB.

Poglavje 10

Sklepne ugotovitve

Uspelo nam je realizirati oba cilja diplomskega dela: raziskati grafne baze in narediti primerjavo med grafnimi in relacijskimi SUPB. Ko smo raziskovali grafne baze smo ugotovili, da gre za kompleksno tematiko. Grafni SUPB so si med seboj zelo različni. Nekateri bazirajo na modelu dokumentnih baz, nekateri na modelu ključ-vrednost ali pa so nekakšen hibrid vsega. Razvijalci Neo4j zase trdijo, da so t. i. nativna grafna baza, drugi pa, npr. ljudje pri SUPB Titan, to demantirajo in razlagajo, da je termin nativen napačen in da nativnih grafnih baz ni. Skratka, materiala za raziskovanje je bilo veliko. Diplomsko delo bi lahko posvetili samo primerjavi grafnih SUPB.

Raziskali smo tudi grafni shranjevalni mehanizem za MySQL imenovan OQGRAPH, ki pa to ni bil (vsaj ne v polni meri). OQGPRAH nam ni bil v nobeno pomoč pri problemu iskanja soigralcev, izkazal pa se je pri iskanju najkrajše poti. Tu vsekakor lahko potegnemo zaključek, da je shranjevalni mehanizem OQGRAPH daleč od nadomestka za grafni SUPB.Primerjava SUPB OrientDB in Neo4j je dala neodločen rezultat. Neo4j in jezik Cypher sta po našem mnenju veliko lažja za uporabo, vendar pa ima OrientDB boljši spletni vmesnik. Neodločen rezultat smo dobili tudi pri časovnih meritvah, vendar pa bi v tem primeru lahko zaključili, da je OrientDB hitrejši, če je poizvedba sestavljena optimalno.

Za zaključek lahko rečemo, da so grafne baze podatkov najbolj primerna

rešitev za grafne podatke in grafne probleme. V primerih, kot smo jih raziskali v diplomskem delu, se relacijske podatkovne baze niso izkazale kot konkurenčne.

Literatura

[1] Neo4j blog. [Online]. Dosegljivo:
http://Neo4j.com/blog/Neo4j-1-0-released/, 18. 4. 2015.

[2] A. Vukotic, N. Watt. *Neo4j in Action*. Manning, 2014.

[3] Graph database. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Graph_database, 20.10.2015.

[4] A Tale of Two Graphs: Property Graphs as RDF in Oracle. [Online].
Dosegljivo:
http://openproceedings.org/EDBT/2014/edbticdt2014industrial_submission_28.pdf,
20.12.2015.

[5] SUPB tipa ključ-vrednost Redis. [Online]. Dosegljivo:
http://redis.io/, 20.1.2016.

[6] SUPB tipa ključ-vrednost Memcached. [Online]. Dosegljivo:
http://memcached.org/, 20.1.2016.

[7] Stolpični SUPB Apache HBase. [Online]. Dosegljivo:
http://hbase.apache.org, 20.1.2016.

[8] Dokumentni SUPB MongoDB. [Online]. Dosegljivo:
http://www.mongodb.org, 20.1.2016.

[9] Dokumentni SUPB Couchbase. [Online]. Dosegljivo:
www.couchbase.com, 20.1.2016.

[10] Dokumentni SUPB Apache CouchDB. [Online]. Dosegljivo: http://couchdb.apache.org, 20.1.2016.

[11] Grafni SUPB Neo4j. [Online]. Dosegljivo: http://Neo4j.com, 20.1.2016.

[12] Grafno-dokumentni SUPB OrientDB. [Online]. Dosegljivo: http://orientdb.com, 20.1.2016.

[13] Grafni SUPB Titan. [Online]. Dosegljivo: http://thinkaurelius.github.io/titan/, 20.1.2016.

[14] Grafni SUPB FlockDB. [Online]. Dosegljivo: https://github.com/twitter/flockdb, 20.1.2016.

[15] What is a Graph Database. [Online]. Dosegljivo: http://neo4j.com/why-graph-databases/, 12.2.2016.

[16] A Letter Regarding Native Graph Databases. [Online]. Dosegljivo: http://thinkaurelius.com/2013/11/01/a-letter-regarding-native-graph-databases/, 12.2.2016

[17] Programski jezik Gremlin. [Online]. Dosegljivo: https://github.com/tinkerpop/gremlin/wiki, 20.12.2015.

[18] Object-relational impedance mistmatch. [Online]. Dosegljivo: http://c2.com/cgi/wiki?ObjectRelationalImpedanceMismatch, 21.1.2016.

[19] ArrangoDB. [Online]. Dosegljivo: http://www.arangodb.com, 23.1.2016.

[20] Allegro Graph. [Online]. Dosegljivo: http://franz.com/agraph/allegrograph, 23.1.2016.

[21] Apache Giraph. [Online]. Dosegljivo: http://giraph.apache.org, 23.1.2016.

[22] GraphBase. [Online]. Dosegljivo:
http://graphbase.net, 23.1.2016.

[23] GraphPack. [Online]. Dosegljivo:
http://code.google.com/p/graphpack, 23.1.2016.

[24] HyperGraphDB. [Online]. Dosegljivo:
http://hypergraphdb.org, 23.1.2016.

[25] InfiniteGraph. [Online]. Dosegljivo:
http://www.objectivity.com/products/infinitegraph, 23.1.2016.

[26] InfoGrid. [Online]. Dosegljivo:
http://infogrid.org/trac, 23.1.2016.

[27] BerkeleyDB. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Berkeley_DB, 23.1.2016.

[28] Apache TinkerPop. [Online]. Dosegljivo:
https://tinkerpop.incubator.apache.org, 23.1.2016.

[29] Blueprints. [Online]. Dosegljivo:
https://github.com/tinkerpop/blueprints/wiki, 23.1.2016.

[30] I. Robinson, J. Webber, E. Eifrem. *Graph Databases*. O'Reilly, 2013.

[31] Dijkstrin algoritem. [Online]. Dosegljivo:
https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html,
18.1.2016.

[32] Podatkovna baza Cassandra. [Online]. Dosegljivo:
http://cassandra.apache.org/, 22.10.2015.

[33] Podatkovna baza FlockDB. [Online]. Dosegljivo:
https://blog.twitter.com/2010/introducing-flockdb, 22.10.2015.

[34] Oraclov prevzem podjetja Sun. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle, 24.11.2015.

[35] Open query graph engine. [Online]. DOsegljivo: http://www.psce.com/blog/2015/03/06/handling-hierarchy-in-mysql-with-oqgraph/, 27.11.2015.

[36] Standard ISO/IEC 9075:2011. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/SQL:2011, 25.11.2015.

[37] Popularnost podatkovnih baz v letu 2015. [Online]. Dosegljivo: http://db-engines.com/en/ranking, 22.11.2015.

[38] Graph or Document API? [Online]. Dosegljivo: http://orientdb.com/docs/last/Choosing-between-Graph-or-Document-API.html, 22.11.2015.

[39] Titan. [Online]. Dosegljivo: https://github.com/thinkaurelius/titan, 26.12.2015.

[40] Način Rest. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Representational_state_transfer, 18.11.2015.

[41] Neo4j JDBC API. [Online]. Dosegljivo: https://github.com/neo4j-contrib/neo4j-jdbc, 12.2.2016.

[42] Vgrajeni način za mySQL. [Online]. Dosegljivo: https://www.mysql.com/oem/, 18.11.2015.

[43] Neo4j Traversal API. [Online]. Dosegljivo: http://Neo4j.com/docs/stable/tutorial-traversal-java-api.html, 24.1.2016.

[44] Neo4j example datasets. [Online]. Dosegljivo: http://Neo4j.com/developer/example-data/, 2.12.2015.

Dodatek A

Poizvedbi v jeziku Cypher za drugi in tretji nivo iskanja soigralcev Toma Hanksa:
Nivo dva:

```
MATCH (tom:Person{name:"Tom Hanks"})-[:ACTS_IN]->(m)
<-[:ACTS_IN]-(fof:Person),
     (fof)-[:ACTS_IN]->(m2)<-[:ACTS_IN]-(fof2:Person)
RETURN COUNT(fof2.name)
```

Nivo tri:

```
MATCH (tom:Person{name:"Tom Hanks"})-[:ACTS\_IN]->(m)
                                  <-[:ACTS\_IN]-(fof:Person),
       (fof)-[:ACTS\_IN]->(m2)<-[:ACTS\_IN]-(fof2:Person),
       (fof2)-[:ACTS\_IN] -> (m3)<-[:ACTS\_IN]-(fof3:Person)
        RETURN COUNT(fof3.name)}
```

Poizvedbi v OrientDB za drugi in tretji nivo iskanja soigralcev Toma Hanksa:
Nivo dva:

```
SELECT COUNT(name) as name FROM
(SELECT expand(out('ACTS_IN').in('ACTS_IN'))
```

```
FROM Movie WHERE in('ACTS_IN').name IN
(SELECT DISTINCT(name) as name FROM (SELECT expand(in('ACTS_IN'))
FROM Movie WHERE in('ACTS_IN').name IN 'Tom Hanks')
WHERE name <> 'Tom Hanks')) WHERE name <> 'Tom Hanks';
```

Nivo tri:

```
SELECT COUNT(name) as name FROM
  (SELECT expand(in('ACTS_IN')) FROM Movie
  WHERE in('ACTS_IN').name IN (SELECT DISTINCT(name) as name
  FROM (SELECT expand(in('ACTS_IN')) FROM Movie
  WHERE in('ACTS_IN').name IN (SELECT DISTINCT(name) as name
  FROM (SELECT expand(in('ACTS_IN')) FROM Movie
  WHERE in('ACTS_IN').name IN 'Tom Hanks')
  WHERE name <> 'Tom Hanks'))
  WHERE name <> 'Tom Hanks'))
WHERE name <> 'Tom Hanks';
```

Poizvedbi v jeziku MySQL za drugi in tretji nivo iskanja soigralcev Toma Hanksa:

Nivo dva:

```
SELECT COUNT(DISTINCT person.id)
FROM person JOIN relation ON person_id = person.id
WHERE movie_id IN
  (SELECT movie_id FROM relation WHERE person_id IN
    (SELECT person.id FROM person JOIN relation
      ON person_id = person.id WHERE movie_id IN
        (SELECT movie_id FROM relation
        WHERE person_id = 31)
      AND person.id != 31 AND type = 'acts_in'))
```

```
    AND person.id != 31 AND type = 'acts_in';
```

Nivo tri:

```
SELECT COUNT(DISTINCT person.id)
FROM person JOIN relation ON person_id = person.id
WHERE movie_id IN
  (SELECT movie_id FROM relation WHERE person_id IN
    (SELECT DISTINCT person.id FROM person
    JOIN relation ON person_id = person.id
    WHERE movie_id IN
     (SELECT movie_id FROM relation WHERE person_id
     IN (SELECT person.id FROM person JOIN
     relation ON person_id = person.id
     WHERE movie_id IN
      (SELECT movie_id FROM relation WHERE person_id = 31)
      AND person.id != 31 AND type = 'acts_in'))
    AND person.id != 31 AND type = 'acts_in'))
AND person.id != 31 AND type = 'acts_in';
```

Poizvedba v jeziku Cypher za iskanje najkrajše poti med igralcem Tomom Hanksom in Melom Gibsonom:

```
MATCH p=shortestPath(
  (tom:Person {name:"Tom Hanks"})-[*]
    -(mel:Person {name:"Mel Gibson"})
)
RETURN p
```

Poizvedba v OrientDB za iskanje najkrajše poti med igralcem Tomom Hanksom in Melom Gibsonom:

```
SELECT shortestPath((SELECT FROM Person WHERE name = "Tom Hanks"),
(SELECT FROM Person WHERE name = "Mel Gibson"), "Both")
```